ON-DEMAND AND INCREMENTAL APPLICATION ADAPTATION

Field of the Invention

The present invention relates to computer software applications and more particularly to adaptation of such applications for operation with a requesting device.

Background

Computer software applications may be accessed by a variety of devices remotely via a network. Generally, such applications require adaptation for operation with the particular devices requesting or accessing the application. Adaptation is typically performed either when designing/implementing or deploying an application. However, an application may be adapted for a particular requesting device "on demand" (i.e., at the time of the actual request). Furthermore, the type of accessing device may be known or unknown to the application, at the time of a request.

For the case of a known type of requesting device, the current approach is to adapt and store the entire application prior to access, which disadvantageously results in significant resource utilization such as processing time and memory.

For the case of unknown device types, the current approach is also to adapt the entire application. In addition to the aforementioned disadvantages associated with known device types, other disadvantages such as an increased response time (user waiting time) and an increased difficulty to support in terms of Quality of Service (QoS) also exist in relation to unknown device types.

U.S. Patent Application No. 2001/0049286A1, entitled "Device registry server for automatic connection and data exchange between pervasive devices and backend systems" was published on December 6, 2001 in the name of Hansmann, et al. and is assigned to International Business Machines (IBM) Corporation. The specification relates to communication improvements between a mobile device and backend system applications. However, a backend application that has already been customised for use with the mobile device is assumed.

A paper by Indulska et al., entitled "Vertical handover based adaption for multimedia applications in pervasive systems", is included in the Proceedings of the

Joint International Workshop on Interactive Distributed Multimedia Systems and Protocols for Multimedia Systems (IDMS/PROMS 2002). However, the form of adaptation described is context-based and uses context information relating to user devices, user location, application requirements and network environment.

A need exists for incremental adaptation of computer software applications in response to user requests.

Summary

According to aspects of the present invention, there are provided a method, an apparatus and a computer program product for on-demand and incremental application adaptation.

Components of an application that may be requested by a user in the future are identified and adapted for operation with a device of the user in response to a request for a component of the application by the user. Optionally, only the requested and identified components are adapted for operation with the requesting device, which may enable component adaptation within a specified maximum time period.

A reachability analysis may be performed to identify components reachable from the requested component and components that are within a specified distance of the requested component may be identified or selected. The identified components may further be selected or identified based on historical request patterns relating to the application.

Brief Description of the Drawings

Embodiments are described hereinafter, by way of example only, with reference to the accompanying drawings in which:

Fig. 1 is a schematic representation of a network environment in which embodiments of the present invention can be practised;

Fig. 2 is a flow graph of a computer software application;

Fig. 3 is a flow diagram of a method for adapting components of a computer software application in accordance with an embodiment of the present invention;

Fig. 4 is a flow diagram of an algorithm for identifying and adapting components of a computer software application in accordance with another embodiment of the present invention; and

Fig. 5 is a schematic representation of a computer system wherewith embodiments of the present invention may be practised.

Detailed Description

A method, an apparatus and a computer program product are described herein for incremental on-demand adaptation of a computer software application. The method, apparatus and computer program product are described with specific reference to applications accessible via the Internet, however, it is not intended that the present invention be so limited as the principles of the present invention have general applicability to other networks such as private or proprietary networks.

Applications accessed by a client should be compatible with specific requirements of the client. The client may comprise a device with hardware requirements such as screen size, screen resolution, image display capability, memory available for an application, CPU speed, bandwidth, etc. Software requirements of a client device may, for example, comprise the markup language supported by a browser hosted on the client (e.g., WML 1.3), client-side scripts if any, and the kind of operating system support available. An application may comprise a generic application. However, other kinds of application may be specifically written for a set of clients/devices having similar capabilities. Such applications need to be adapted to meet the client/device requirements. The components of the application that are processed on the device side and the data sent to the device need to be adapted.

For example, consider a web application developed for a generic client. The application comprises two types of components – the view (e.g., HTML or WML pages), which is written in a markup language to perform user interaction, and the component that contains data and/or business logic (e.g., java bean). The web application is deployed on the server side. When a client accesses the application, a view is sent to the client based on the user input. Each view acts as a node in an application flow graph and the java bean invocations act as edges from one node to

the other. Since the view is displayed on the client side, the view needs to be in a markup language supported by the device browser and must meet the memory limit. Any images that the device browser cannot display or that result in a page size greater than the memory limit must be discarded. Thus, the views for the web applications on the server side need to be adapted to the requirements of each client/device.

Fig. 1 shows a network environment in which embodiments of the present invention can be practised. A remote device 110 accesses a computer software application 140 via the Internet 120 and a server 130. The application 140 may be hosted on the server 130 or on a further server (not shown). The remote device 110 may comprise a computing device such as a personal computer (PC), a personal digital assistant (PDA), a mobile telephone, or any other commercially available or proprietary computing device. Modifications to the network environment of Fig. 1, as well as alternative network environments are also suitable for practising embodiments of the invention, as would be well understood by a person skilled in the art. For example, the network is not restricted to the single remote device 110, but may comprise numerous remote devices for accessing the application 140 that are similar or dissimilar to the remote device 110.

Fig. 2 shows a flow graph 200 of a computer software application, such as application 140 in Fig. 1, in which each vertex denotes a component of the application 140 and each edge denotes a transition between components. For example, the components of the application 140 may comprise pages available on an Internet website. A transition may thus represent a selection of a particular component (page) by a user of the remote device 110. Each component that is accessed by a user of the remote device 110 requires adaptation for use by the remote device 110. Similarly, each component that is accessed by a remote device that is different to the remote device 110 requires adaptation for use by that remote device.

Fig. 3 shows a method for adaptation of components of a computer software application. A request from a device operated by a user of the application is received at step 310. Components of the application that may be requested by the user are identified, at step 320, in response to the request. At step 330, the identified components are adapted for operation with the requesting device.

The identified components may be adapted within a specified maximum time period to comply with Quality of Service (QoS) constraints. The components for adaptation may be identified based on a representation of possible control flow paths through the application that can be selected by a user. Components of the application may comprise Internet webpages selected by the user. Components of the application that are not likely to be requested by the user are preferably not adapted to conserve system resources and improve system response time.

Various methods may be used to identify components of an application for adaptation based on a component currently selected by a user. Examples of such methods comprise graph analysis and methods based on statistical algorithms, learning algorithms, and response-time algorithms.

Graph Analysis

Graph analysis can be used to identify subgraphs in a dependency graph. Each subgraph is then adapted in its entirety (i.e., including articulation points, bridges, loops, prefetching techniques, etc.). For example, a reachability analysis is performed to determine the set of components that are reachable from the current component. Then, only the reachable components are adapted.

For purposes of the present disclosure, the graph comprises an application flow graph. An articulation point is a node apart from the root node that, when removed, breaks the graph into more than one component. A bridge comprises an edge of the graph, whose removal causes the graph to become disconnected. A loop comprises a cycle in the graph.

Referring to the flow graph 200 of Fig. 2, the component 270 is unreachable from the component 230. The components 230, 250 and 240 are exclusive of each other; that is, each of the components 230, 250 and 240 cannot be reached from each other. The component 280 is likely to be requested by a user from the component 230. The component 270 need not be adapted when the component 230 is the current component since the component 270 is unreachable from the component 230. The component 290 can always be adapted since this component is likely to be visited from all paths of the flow graph 200.

Statistical

The statistical method is based on patterns of components previously requested or selected by users. Only the components most likely to be selected or requested by a user are adapted for the user's particular device. For this method, it is assumed that historical patterns of user traversals through a flow graph are stored. Then, the probability of user selection of each component reachable from a current component is computed. Assuming that a component (b) is reachable from a component (a), the probability of selection of component (b) from component (a) is determined according to a Bayesian method as follows:

$$\text{probability}(b|a) = \frac{\text{Number of paths in which "a-b" occurs}}{\text{Number of paths in which "a-?" occurs}}$$

As an example, suppose that the component 250 is more often followed by the component 260 than the component 270 in user traversals of the application represented by Fig. 2. In accordance with the statistical method, the component 260 should be adapted earlier than the component 270. Moreover, the component 270 need not be adapted until specifically requested by a user.

Learning

In the learning method, an embedded learning algorithm learns from historical request patterns and previously incorrect predictions to dynamically identify the most likely components to be requested. The method may be implemented using a reinforcement learning algorithm that associates a penalty with wrong prediction and biases the probability of selection of successive components more towards recent historical patterns than older historical patterns.

As an example using the flow graph 200 in Fig. 2, it is possible that although the component 250 may have most often been followed by the component 260 historically, the component 250 has more recently been followed by the component 270, particularly for a user U. The probability of selection of the component 270 from the component 250 will thus be higher than the probability of selection of the component 260 from the component 250, particularly for the user U. Consequently, the priority for adaptation of the component 270 should be higher than the priority for adaptation of the component 260 for the user U.

Response-time

In this method, the maximum number of components in the neighbourhood of the current component that can be adapted within a desired response-time are adapted. Firstly, the times t_i required to adapt each respective component C_i are calculated. Then, given a maximum response time T and starting from the current component, the times t_i are added in BFS (Breadth First Search) order until the sum of the added times is less than, but as close as possible to, the maximum response time T. A BFS order is the order of traversing a graph such that when a particular node is visited, then all the neighbour nodes of that node are also visited.

Embodiment Using Graph Analysis

Fig. 4 shows a method for identifying and adapting components of an application using graph analysis.

A request for a component of the application is received from a device at step 410. If a particular component is not specified, the starting component of the application is selected as the requested component. At step 420, components reachable from the requested component are identified by performing a reachability analysis. A determination is made at step 430 whether the lookahead distance is fixed or variable. A lookahead distance comprises an integer value greater than or equal to one and is representative of a number of transitions between two components. For example, the distance between the components 210 and 240 in Fig. 2 is 1, the distance between the components 210 and 280 is 2, and the distance between the components 220 and 290 is 3. In cases where different lookahead distances are applied to different components, a variable lookahead distance k_i may be determined for each component c_i. The distances k and k_i may be determined in accordance with Quality of Service (QoS) constraints. Factors for determination may comprise the amount of time necessary for adapting one or more components and/or how long a user takes to respond to a current request, if at all. A high value of lookahead distance is generally desirable, particularly where resource capacity is available. However, too high a value of distance k may result in unacceptably long response times for a user.

If the lookahead distance is fixed, reachable components within a fixed distance k of the currently requested component are selected and processing continues at step

470. If the lookahead distance is variable, a variable distance k_i is determined for the currently requested component c_i, at step 450. Reachable components within the distance k_i of the currently requested component c_i are selected, at step 460. Thereafter, the components selected in step 440 or step 460 are adapted for operation with the requesting device, at step 470. Any of the components already adapted for operation with the requesting device need not be adapted again.

Referring to the flow graph 200 of Fig. 2, assume that that the distance k = 1 (by selection) and that the components of the flow graph 200 have not yet been adapted to a particular device or set of devices having common characteristics pertinent to the adaptation. Upon a request from a user, the component 210 is adapted. During or after adaptation of the component 210, the components 220, 230 and 240 are identified to be reachable within the distance k = 1 from the component 210. Components requiring adaptation can be identified during adaptation of another component by execution of an algorithm in multiple threads. Upon completion of adaptation of the component 210, the components 220, 230 and 240 are adapted. If the user's next request is for the component 230, only the component 280 will be identified for adaptation on account of being within the distance k = 1 from the component 230. However, if the user's next request is for the component 220, only the component 250 will be identified for adaptation on account of being within the distance k = 1 from the component 220.

Computer hardware and software

Fig. 5 is a schematic representation of a computer system 500 that can be used to practise the methods described herein. Specifically, the computer system 500 is provided for executing computer software that is programmed to assist in performing a method for adaptation of a computer software application. The computer software executes under an operating system such as MS Windows XP™ or Linux™ installed on the computer system 500.

The computer software involves a set of programmed logic instructions that may be executed by the computer system 500 for instructing the computer system 500 to perform predetermined functions specified by those instructions. The computer software may be expressed or recorded in any language, code or notation that

comprises a set of instructions intended to cause a compatible information processing system to perform particular functions, either directly or after conversion to another language, code or notation.

The computer software program comprises statements in a computer language. The computer program may be processed using a compiler into a binary format suitable for execution by the operating system. The computer program is programmed in a manner that involves various software components, or code means, that perform particular steps of the methods described hereinbefore.

The components of the computer system 500 comprise: a computer 520, input devices 510, 515 and a video display 590. The computer 520 comprises: a processing unit 540, a memory unit 550, an input/output (I/O) interface 560, a communications interface 565, a video interface 545, and a storage device 555. The computer 520 may comprise more than one of any of the foregoing units, interfaces, and devices.

The processing unit 540 may comprise one or more processors that execute the operating system and the computer software executing under the operating system. The memory unit 550 may comprise random access memory (RAM), read-only memory (ROM), flash memory and/or any other type of memory known in the art for use under direction of the processing unit 540.

The video interface 545 is connected to the video display 590 and provides video signals for display on the video display 590. User input to operate the computer 520 is provided via the input devices 510 and 515, comprising a keyboard and a mouse, respectively. The storage device 555 may comprise a disk drive or any other suitable non-volatile storage medium.

Each of the components of the computer 520 is connected to a bus 630 that comprises data, address, and control buses, to allow the components to communicate with each other via the bus 530.

The computer system 500 may be connected to one or more other similar computers via the communications interface 565 using a communication channel 585 to a network 580, represented as the Internet.

The computer software program may be provided as a computer program product, and recorded on a portable storage medium. In this case, the computer

software program is accessible by the computer system 500 from the storage device 555. Alternatively, the computer software may be accessible directly from the network 580 by the computer 520. In either case, a user can interact with the computer system 500 using the keyboard 510 and mouse 515 to operate the programmed computer software executing on the computer 520.

The computer system 500 has been described for illustrative purposes. Accordingly, the foregoing description relates to an example of a particular type of computer system suitable for practising the methods and computer program products described hereinbefore. Other configurations or types of computer systems can be equally well used to practise the methods and computer program products described hereinbefore, as would be readily understood by persons skilled in the art. For example, the methods and computer program products described hereinbefore can be practised using a handheld computer such as a Personal Digital Assistant (PDA) or a mobile telephone.

Conclusion

A method, an apparatus and a computer software product have been described hereinbefore for selectively adapting components of a computer software application. Advantageously, such components need only be adapted as and when required, thus enabling better utilization of computing resources and an improved Quality of Service (QoS) to users.

The foregoing detailed description provides exemplary embodiments only, and is not intended to limit the scope, applicability or configurations of the invention. Rather, the description of the exemplary embodiments provides those skilled in the art with enabling descriptions for implementing an embodiment of the invention. Various changes may be made in the function and arrangement of elements without departing from the spirit and scope of the invention as set forth in the claims hereinafter.